



H.E.B. ECOLE SUPERIEUR D'INFORMATIQUE

LABORATOIRE DE C++ : PROJET 2

Starlight

Auteurs :
Paul KRIWIN
Simon PLACENTINO

Titulaire du cours :
Dr. Romain ABSIL

20 avril 2015

Table des matières

1	Introduction	3
2	Les classes	4
2.1	Les objets géométriques	4
2.1.1	Ellipse	4
2.1.2	Droite	5
2.1.3	Rectangle	5
2.1.4	Point	6
2.1.5	Utilitaire	6
2.2	Les éléments	7
2.2.1	Element	7
2.2.2	Cristal	8
2.2.3	Destination	8
2.2.4	Lentille	8
2.2.5	Miroir	9
2.2.6	Bombe	9
2.2.7	Rayon	9
2.2.8	Source	9
2.2.9	Mur	9
2.2.10	Niveau	9
2.2.11	Createur de niveau	9
2.3	L'exception	9
2.3.1	Exception Starlight	9
2.4	Les objets visuels	10
3	Structure du programme	11
4	Algorithmes	12
4.1	Réflexion	12
4.2	Intersection	12
4.2.1	Deux droites	12
4.2.2	Droite et rectangle	12
4.2.3	Droite et ellipse	12

5	Test effectués	13
6	Conclusion	14
A	Références	15

Chapitre 1

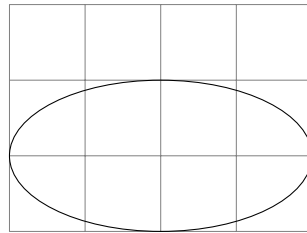
Introduction

Chapitre 2

Présentation succincte des classes

2.1 geometry

2.1.1 ellipse.hpp



Une ellipse est un objet géométrique à deux dimensions représentée par une courbe plane fermée obtenu par découpe d'un cône sur un plan. Si ce dernier est perpendiculaire à l'axe du cône, l'ellipse sera alors un cercle. Éléments caractéristiques d'une ellipse :

- une coordonnée cartésienne de son centre,
- une distance séparant le centre de l'intersection avec une parallèle à l'axe des ordonnées tangente à l'ellipse voulue,
- une distance séparant le centre de l'intersection avec une parallèle à l'axe des abscisse tangente à l'ellipse voulue.

Ces éléments nous permettront de tracer une ellipse selon cette équation :

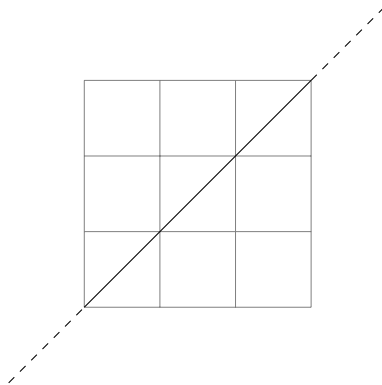
$$\frac{(x - c_x)^2}{x_{radius}^2} + \frac{(y - c_y)^2}{y_{radius}^2} = 1$$

Cette classe peut tout à fait être instancié en objet géométrique elliptique et possède des méthodes d'interactions avec une droite.

```
Ellipse :: getIntersectionsPoints ( Line );
```

Dans le contexte présent, certains éléments du jeu seront des ellipse par le phénomène d'héritage mis en place dans le paradigme orienté objet de C++.

2.1.2 line.hpp



Une droite est une ligne sans épaisseur, rectiligne et infinie dans le plan. Pour exister, une droite aura besoin :

- d'un coefficient angulaire $m = \frac{\Delta y}{\Delta x}$ représentant la distance à parcourir sur l'axe des ordonnées pour une unité de distance sur l'axe des abscisses.
- d'un terme indépendant $p = \frac{y}{m \cdot x}$ représentant le décalage de chaque point sur l'axe des ordonnées,
- ou de deux points de coordonnées dans le plan,
- ou d'un point de coordonnées dans le plan et d'un coefficient angulaire.

Ces éléments nous permettent de tracer une droite selon cette équation :

$$y = m \cdot x + p$$

2.1.3 rectangle.hpp



Un rectangle est une forme géométrique à 4 segments de droite¹ parallèle deux à deux. Ceux-ci vont donc former 4 angles droit ($\frac{\pi}{2}rad$) Cette forme peut être représentée par :

- la coordonnée du coin supérieur gauche $Sg = (x, y)$
- la grandeur des deux segments formant un angle de $\frac{\pi}{2}rad$ en ce point *hauteur* et *largeur*.

1. Un segment de droite est une partie de droite délimitée par deux points non confondus

Ainsi, il sera aisé de déterminer la position des autres coins

- $Sd = (Sg_x + largeur, Sg_y)$
- $Ig = (Sg_x, Sg_y + hauteur)$
- $Id = (Sg_x + largeur, Sg_y + hauteur)$

Il existe plusieurs cas particuliers de droite :

- une droite parallèle à l’axe des ordonnées qui aura pour équation

$$x = a, a \in \mathbb{R},$$

- une droite parallèle à l’axe des abscisses qui aura pour équation

$$y = b, b \in \mathbb{R},$$

2.1.4 point.hpp

Un point est un objet mathématique permettant de situer un element dans un plan ou dans l’espace. Dans notre cas, plus spécifiquement dans un plan à deux dimensions. Celui-ci peut-être représenté de plusieurs manières dans le plan cartésien² :

- sous la forme d’une coordonnées cartésienne à l’aide de
 - une origine,
 - deux vecteurs partant de cette origine et perpendiculaires,

$$P = (\vec{x}, \vec{y})$$

- et sous la forme d’une coordonnée polaire à l’aide de
 - une origine,
 - une coordonnée radiale r ,
 - une coordonnée angulaire α .

2.1.5 utilities.hpp

Le namespace “utilities” mis en place ici est un ensemble de fonctions et valeurs constantes spécifiquement définies pour les calculs intervenant dans le projet.

constantes :

PI est une approximation de π sur 26 décimales,

PI_2 est une approximation de $\frac{\pi}{2}$ sur 26 décimales,

PI_4 est une approximation de $\frac{\pi}{4}$ sur 26 décimales,

EPSILON est une marge d’erreur de 10^{-7} ,

INF représente un nombre dit “infini” dans le milieu informatique.

2. <http://www.cslaval.qc.ca/sitsatl11/maths2003/cartesien.html>

fonctions :

Resolution d'équation du second degre

```
utilitaire :: secondDegreeEquationSolver
```

Transforme un angle exprime en radian en un angle exprime en degres

```
utilitaire :: angleAsDegree
```

Permet de tester l'egalite ou l'inegalite entre deux nombre reels a un Epsilon d'erreur

```
utilitaire :: equals  
utilitaire :: greaterOrEquals  
utilitaire :: lessOrEquals
```

Permet de trouver le coefficient angulaire a partir de deux points"

```
utilitaire :: slopeFromPoints
```

Permet de trouver la valeur tangente d'un angle en radian mais aussi de retourner une valeur particuliere pour la tangente de $\pi/2$

```
utilitaire :: tan
```

Permet de savoir si α vaut $\frac{\pi}{2} + n * \pi$ $n \in \mathbb{N}$

```
utilitaire :: isHalfPiPlusNPi
```

2.2 elements

2.2.1 element.hpp

Cette classe, abstraite et donc non instanciable, représente un élément du jeu lié à un, et un seul, niveau du jeu. Cette classe permet donc d'établir une communication entre les différents éléments du niveau et le niveau lui-même à travers un référencement de ce dernier. Par le phénomène d'héritage, chaque élément pourra établir sa manière propre de réagir avec le niveau en lui exprimant :

- ses points d'intersection avec un rayon si il lui sont demandés,

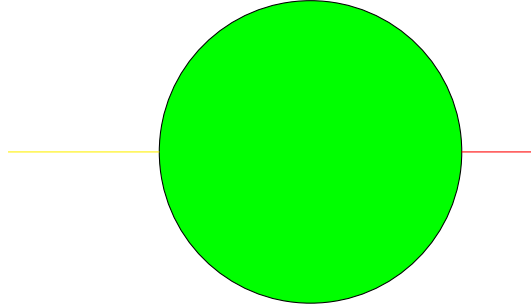
```
Element :: includeRay (Ray) ;
```

- les actions à effectuer si un contact avec le rayon a eu lieu.

```
Element :: reactToRay (Ray) ;
```

Il sera donc aisé pour le niveau de gérer les collisions des éléments de manière anonyme et optimale (cf Niveau).

2.2.2 crystal.hpp

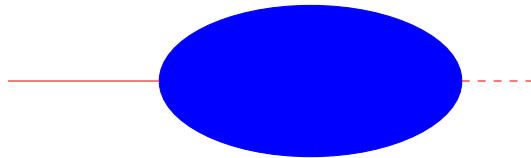


Cette classe est, par héritage, un élément ainsi qu'une ellipse. Il peut donc être modéliser graphiquement, entant que cette dernière, dans le plan et communique avec le niveau auquel il est lié. Le cristal modifie la longueur d'onde d'un rayon qui le traverse. D'un point de vue technique, il en relève donc d'un choix personnel, il ne va pas amplifier le rayon entrant à partir de sa sortie mais arrêtera ce rayon et en créera un nouveau qui, lui, aura subit l'action du cristal.

2.2.3 dest.hpp

Cette classe est, par héritage, un élément ainsi qu'un rectangle. Il peut donc être modéliser graphiquement, entant que cette dernière, dans le plan et communique avec le niveau auquel il est lié. La destination est le but du jeu et l'atteindre terminera automatiquement la partie sur une victoire.

2.2.4 lens.hpp



Cette classe est, par héritage, un élément ainsi qu'un rectangle. Il peut donc être modéliser graphiquement, entant que cette dernière, dans le plan et communique avec le niveau auquel il est lié. La lentille est un éventuel obstacle à un rayon puisqu'il ne le laissera passer que si la longueur d'onde de ce dernier respecte le critère

$$lens_{min} \leq ray_{\alpha} \leq lens_{max}$$

Deux issues sont alors possible :

- la longueur d'onde entre dans l'intervall et le rayon est renouvelé après la lentille
- ou elle ne rentre pas dans l'intervall défini et le rayon sera terminé par la lentille

et ce même si le rayon tiré est tangent à la lentille. En effet, si la droite représentant le rayon est tangent à l'ellipse représentant la lentille c'est qu'un point d'intersection existe entre les deux. Il est donc de ce point d'agir comme l'aurait fait l'entière lentille à tout autre point même si cette réaction particulière peut être discutable.

2.2.5 mirror.hpp

Cette classe est, par héritage, un élément ainsi qu'une droite. Le miroir peut donc être modéliser graphiquement, entant que cette dernière, dans le plan et communiquer avec le niveau auquel il est lié. Le miroir est, plus précisément, un segment de droite délimité par deux points. Puisque le miroir effectue une rotation autour d'un point $p \in \text{miroir}$, il n'est pas directement délimité par ses deux extrémités mais bien par sa longueur, son angle, la position absolue (cartésienne) et relative (distance par rapport à une extrémité) de son point de rotation qui définissent, dynamiquement, ses points délimiteurs. Pour trouver les deux points il suffit de :

1. extrémité gauche = $(x_{pivot} - \text{position absolue}, y_{pivot})$
2. extrémité droite = $(x_{gauche} + \text{taille du segment}, y_{pivot})$
3. rotation des deux points autour du pivot

2.2.6 nuke.hpp

2.2.7 ray.hpp

2.2.8 source.hpp

2.2.9 wall.hpp

2.2.10 level.hpp

2.2.11 levelfactory.hpp

2.3 exception

2.3.1 starlightexception.hpp

Il est nécessaire, pour bon nombre des classes créées, de valider les arguments passés en paramètre dans le but de ne pas produire d'objets incohérents par rapport à l'analyse préalable du travail à fournir. Pour ce faire, des exceptions doivent être levées quand une instantiation créera un objet non désiré. Cette classe hérite de `std::exception` appartenant à la librairie standard. Elle n'a aucune capacité supplémentaire mise à part être spécifique à ce projet.

Les différentes classes pouvant lever cette exception sont :

crystal si la taille de son rayon ne lui permet pas d'exister dans le plan,
lens si son intervalle de longueur d'onde n'est pas cohérent,
level si ses dimensions ne lui permettent pas d'exister dans le plan,
mirror si ses dimensions ne lui permettent pas d'exister dans le plan, si sa position ou son angle n'entre pas dans les limites imposées,
nuke si la taille de son rayon ne lui permet pas d'exister dans le plan,
ray si sa longueur d'onde n'entre pas dans l'intervalle cohérent imposé,
source si sa longueur d'onde n'entre pas dans l'intervalle cohérent imposé,
wall si ses points déterminants ne lui permettent pas d'exister dans le plan,
ellipse si ses dimensions ne lui permettent pas d'exister dans le plan,
rectangle si ses dimensions ne lui permettent pas d'exister dans le plan.

2.4 view

Chapitre 3

Structure générale du programme

Chapitre 4

Détail des algorithmes utilisés

4.1 Algorithme de réflexion

4.2 Algorithme d'intersection

4.2.1 Intersection de deux droites

4.2.2 Intersection d'une droite et d'un rectangle

4.2.3 Intersection d'une droite et d'une ellipse

Chapitre 5

Test effectués

Chapitre 6

Conclusion

Annexe A

Références